

## CLAIMS

1. A method of enabling execution of data-relative code within a non data-relative environment, including the steps of:
  - i) locating one or more instances of the use of a data-relative offset within a module of a code segment;
  - ii) calculating a new offset independent of a data segment; and
  - iii) replacing the data-relative offset with the new offset in the code segment module.
2. A method as claimed in claim 1 wherein the data-relative offset is used for computing the target address of a branch instruction.
3. A method as claimed in claim 2 wherein the new offset is relative to the instruction pointer.
4. A method as claimed in claim 3 wherein the new offset is calculated using the formula:
$$\text{new\_offset} = \text{link\_time\_data\_address} + \text{data\_relative\_offset} - (\text{link\_time\_module\_start} + \text{instruction\_pointer\_offset});$$
wherein:
  - new\_offset = the new offset;
  - link\_time\_data\_address = the data address during link time;
  - data\_relative\_offset = the address relative to the data address;
  - link\_time\_module\_start = the address of the start of the module of the code segment during link time; and
  - instruction\_pointer\_offset = the offset, relative to the start of the code segment module, of the instruction that calculates the new address using the instruction pointer (IP).
5. A method as claimed in claim 3 wherein steps (i) to (iii) occur during runtime of an application using the code segment.
6. A method as claimed in claim 5, including the step of:

iv) executing the modified code segment module.

5 7. A method as claimed in claim 6 wherein the code segment module is executed on an HP-UX platform.

8. A method as claimed in claim 6 wherein the code segment is compiled for a little-endian system and the code segment module is executed on a big-endian system.

10 9. A method as claimed in claim 6 wherein the code segment module is executed on a non-native platform.

15 10. A method as claimed in claim 5 wherein the new offset is an absolute runtime address.

11. A method as claimed in claim 10 wherein the absolute runtime address is calculated using the formula:  
runtime\_addr = link\_time\_data\_addr + data\_relative\_offset –  
20 link\_time\_code\_segment\_start + runtime\_code\_segment\_start;  
wherein:  
runtime\_addr = the absolute runtime address;  
link\_time\_data\_addr = the data address during link time;  
data\_relative\_offset = the address relative to the data address;  
25 link\_time\_code\_segment\_start = the address of the start of the code segment during link time; and  
run\_time\_code\_segment\_start = the absolute address of the start of the code segment during runtime.

30 12. A method as claimed in claim 2 wherein steps (i) to (iii) occur outside of the runtime of the application using the code segment.

13. A method as claimed in claim 12, including the step of:  
v) saving the modified code segment module to non-volatile memory.

14. A method as claimed in claim 1 wherein the code segment module is within a shared library.
- 5 15. A method as claimed in claim 14, including the step of:  
vi) copying the code segment module to modifiable memory;  
wherein step (vi) occurs before step (iii) and wherein step (iii) the data-  
relative offset is replaced in the copied code segment module.
- 10 16. A method as claimed in claim 15, including the step of:  
vii) allocating the modifiable memory read, write and execute  
permissions;  
wherein step (vii) occurs before step (vi).
- 15 17. A method as claimed in claim 14 wherein the code segment within the shared  
library is mapped as writable.
18. A method as claimed in claim 14 wherein the code segment module is one  
selected from the set of an .init section from the shared library and a .fini section  
20 from the shared library.
19. A method as claimed in claim 1 wherein the code segment is within an  
application.
- 25 20. A method as claimed in claim 1 wherein the code segment was compiled using  
a compiler which inserts code that uses data-relative offsets.
21. A method as claimed in claim 20 wherein the code segment was compiled using  
gcc 2.96.
- 30 22. A method as claimed in claim 1 wherein a dynamic loader performs all the  
steps.

23. A method as claimed in claim 2 wherein the data-relative offset is located in step (i) by backtracing the target register of the branch instruction.
24. A method as claimed in claim 1 wherein the code segment is a Linux code segment.
25. A system for enabling execution of data-relative code within a non data-relative environment, including:
- i) A processor adapted to locate the use of data-relative offsets within a module of a code segment, to calculate a new offset independent of a data segment, and to replace the data-relative offset with the new offset in the code segment module; and
  - ii) Memory adapted to store the code segment module.
26. A system as claimed in claim 25 wherein the data-relative offset is used for computing the target address of a branch instruction.
27. A system as claimed in claim 26 wherein the new offset is relative to the instruction pointer.
28. A system as claimed in claim 27 wherein the new offset is calculated using the formula:
- $$\text{new\_offset} = \text{link\_time\_data\_address} + \text{data\_relative\_offset} - (\text{link\_time\_module\_start} + \text{instruction\_pointer\_offset});$$
- wherein:
- new\_offset = the new offset;
  - link\_time\_data\_address = the data address during link time;
  - data\_relative\_offset = the address relative to the data address;
  - link\_time\_module\_start = the address of the start of the module of the code segment during link time; and
  - instruction\_pointer\_offset = the offset, relative to the start of the code segment module, of the instruction that calculates the new address using the instruction pointer (IP).

29. A system as claimed in claim 25 wherein the processor is further adapted to execute an application using the modified code segment module.
30. A system as claimed in claim 29 wherein the new offset is an absolute runtime address.
31. A system as claimed in claim 30 wherein the absolute runtime address is calculated using the formula:  
$$\text{runtime\_addr} = \text{link\_time\_data\_addr} + \text{data\_relative\_offset} - \text{link\_time\_code\_segment\_start} + \text{runtime\_code\_segment\_start};$$
  
wherein:  
runtime\_addr = the absolute runtime address;  
link\_time\_data\_addr = the data address during link time;  
data\_relative\_offset = the address relative to the data address;  
link\_time\_code\_segment\_start = the address of the start of the code segment during link time; and  
run\_time\_code\_segment\_start = the absolute address of the start of the code segment during runtime.
32. A system as claimed in claim 25, including:  
iii) non-volatile memory adapted to store the modified code segment module.
33. A system as claimed in claim 25 wherein the code segment is within a shared library.
34. A system as claimed in claim 33, including:  
iv) modifiable memory adapted to stored the code segment module;  
and wherein the processor is further adapted to copy the code segment module to the modifiable memory and wherein the data-relative offset is replaced in the copied code segment module.
35. A system as claimed in claim 34 wherein the processor is further adapted to allocate the modifiable memory read, write and execute permissions.

36. A system as claimed in claim 33 wherein the code segment module is one selected from the set of an .init section from the shared library and a .fini section from the shared library.
- 5
37. A system as claimed in claim 25 wherein the code segment is within an application.
38. A system as claimed in claim 25 wherein the code segment was compiled using a compiler which inserts code that uses data-relative offsets.
- 10
39. A system as claimed in claim 38 wherein the code segment was compiled using gcc 2.96.
40. A system as claimed in claim 25 wherein the code segment is a Linux code segment.
- 15
41. A system as claimed in claim 29 wherein the code segment module is executed on an HP-UX platform.
- 20
42. A system as claimed in claim 29 wherein the code segment is compiled for a little-endian system and the code segment module is executed on a big-endian system.
43. A system as claimed in claim 29 wherein the code segment module is executed on a non-native platform.
- 25
44. A code segment module modified by the method of claim 1.
45. A binary file containing a code segment module modified by the method of claim 1.
- 30
46. Software for effecting the method of claim 1.

47. Storage media including the software as claimed in claim 46.

48. A computer system for effecting the method of claim 1.

5 49. A memory storing the software of claim 46.